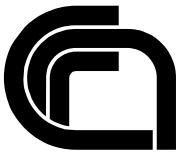


Consiglio Nazionale delle Ricerche

CAMEO: a novel Context-Aware MiddlEware for Opportunistic Mobile Social Networks

V. Arnaboldi, M. Conti, F. Delmastro

IIT TR-02/2012

Technical report

Marzo 2012

Istituto di Informatica e Telematica

CAMEO: a novel Context-Aware MiddlEware for Opportunistic Mobile Social Networks

[Technical Report]

Valerio Arnaboldi, Marco Conti, Franca Delmastro

IIT Institute, National Research Council of Italy
via G. Moruzzi, 1 - 56124 Pisa, Italy
email: firstname.lastname@iit.cnr.it

Abstract

Mobile systems are characterized by several dynamic components like users' mobility, devices' interoperability and interactions among users and their devices. In this scenario context-awareness and the emerging concept of social-awareness become a fundamental requirement to develop optimized systems and applications. In this paper we focus on a novel category of pervasive and mobile systems designed to implement new services and applications that improve users' social interactions through the use of mobile devices. Specifically, we present CAMEO, a light-weight context-aware middleware platform designed to support the development of real-time Mobile Social Networks (MSN). MSN extend the paradigm of Online Social Networks with additional interaction opportunities generated by users' mobility and opportunistic wireless communications. MSN are aimed at creating new and on-demand virtual social networks among mobile users which share interests, habits and needs by providing own-generated contents and without requiring a stable Internet connection. CAMEO provides to MSN applications a set of optimized services for the collection and elaboration of multidimensional context information in addition to efficient opportunistic communication protocols. A prototype of CAMEO has been implemented on Android platform.

Keywords: mobile social networks, middleware, context-aware, social-aware, opportunistic networks

1 Introduction

Context-awareness is a fundamental requirement in the design of mobile and pervasive computing systems. These systems are characterized by a high dynamism mainly due to users' mobility, mobile devices' interoperability, applications' interaction with the external environment and, last but not least, interactions among users and devices. The system must be able to rapidly adapt to these changing conditions and satisfy as much as possible the user's requirements. In

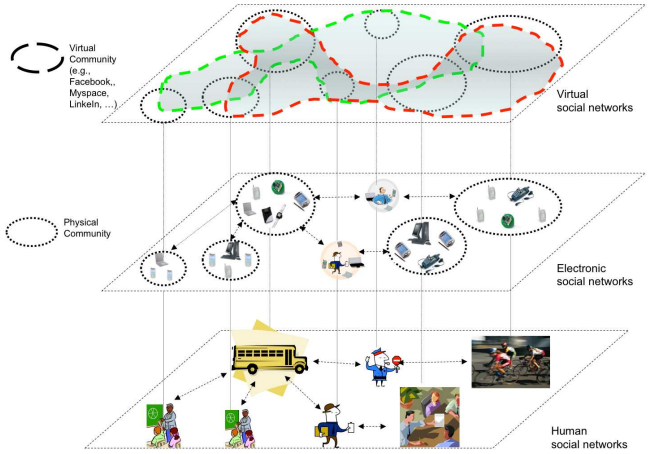


Figure 1: Human, Physical and Virtual Social Networks.

this scenario the use of context information can essentially contribute to the optimization of system's functionalities.

In this paper we focus on a novel category of pervasive and mobile systems designed to implement new services and applications that improve users' social interactions through the use of mobile devices. Currently, with the increasing success of social networking applications and platforms, mobile users are looking for systems which are able to connect them with the others, in order to share user-generated contents to stimulate discussions on personal opinions or share experiences. They would like to be connected anywhere and anytime, through their personal mobile devices, as witnessed by the success of Online Social Network (OSN) applications and their mobile extensions (more than 100 million people actively using Facebook from their mobile devices[1]). In OSN users' interactions are generally based on existing social relationships, and they are mainly driven by common interests, creating thus virtual networks of *known users* (i.e., friends or friends of friends). In fact, in this case users' interactions occur in the virtual world only.

In OSN, users' interactions occur in the virtual world only. By extending this scenario with the interaction opportunities generated by users' mobility and the emerging success of opportunistic wireless communications [9, 19], we can observe a convergence of virtual and physical social networks (i.e., network of contacts among users' devices while users move in the physical space) [14]. In this case users and devices that occasionally encounter in a physical location can automatically discover their own common interests and directly generate and share content with a peer-to-peer communication, without accessing the Internet, creating thus a real-time *mobile social network* (MSN). In this way the network of devices becomes a proxy of the networks of their human owners. MSN are both *people-centric*, mapping human social interactions on the physical network of electronic devices, and *content-centric*, providing efficient services for content dissemination mainly following users' interests. Figure 1 shows a representation of human, physical and virtual social networks.

[1] http://blog.facebook.com/blog.php?post=297879717130

This scenario paves the way to the definition of a novel set of mobile applications designed to improve users' social experiences by enlarging human interactions through the use of mobile devices.

Let's show a practical example. Currently tourists are more and more autonomous in planning their trips and sharing their experiences through the Web, but they need to search in advance for general information on dedicated websites (e.g., www.tripadvisor.com) or through their social network, in case some of their friends has useful information on that topic. In this way they can access information that has been generated in the past and which is typically not tailored on the current user context. Simply moving around the city, they can encounter people that have just visited interesting attractions and that can provide them useful information not available on the Internet (e.g., "At 3PM there was 2 hours queue to visit Colosseum", or "Yesterday dinner at that restaurant X was awful!"). A context-aware dissemination of this kind of information can help interested users to re-schedule their visit, further optimizing their time and avoiding unpleasant experiences.

Mobile Social Networks can also assume an important role in the definition of novel solutions for resource/information sharing in mobile systems. Let's consider the emerging trend of Participatory Sensing applications [6]. Users generate contents related to air pollution, traffic monitoring, risky areas in a participatory fashion by exploiting their own resources (e.g., embedded sensors and camera) and then data is stored on a web server to be shared with others. These applications are typically web-based and exploit the single user with his smartphone as a sensor node to collect the data from the environment. In this way they neglect possibilities offered by the direct interactions among mobile users and their devices. Opportunistic communications and MSN can further enrich participatory sensing applications through the cooperative use of resources belonging to physically connected devices (i.e., devices that are within the same wireless communication range). For example, if node A measures a temperature of 30 °C but it is not able to measure the environmental humidity, it can ask this data to node B, which has the humidity sensor embedded in the smartphone. Then node A can locally correlate the collected information. In the same way a node can delegate to another node the computation of a complex operation or the collection and elaboration of data provided by external sources unreachable from the local device. In this case the MSN offers an opportunistic computing service [15].

Several other application domains can be improved by exploiting MSN: from environmental monitoring to personal safety by generating a real-time network of social support for risky situations (e.g., alert generated by a Personal Health System and forwarded to the MSN, advertisement of polluted city areas for asthmatic patients). In all these scenarios the main target is to improve users' experiences encouraging users' and devices' interactions aimed at sharing useful information/resources for activities organization and life management in general. To this aim it is essential to exploit multi-dimensional context information (including users' profile, generated contents, social interactions and information related to the surrounding environment) in order to provide highly personalized, efficient and effective services in a really dynamic environment.

In this paper we present CAMEO, a light-weight context-aware middleware platform designed to support the development of MSN applications on top of opportunistic networks. In Section 2 we provide a detailed definition of context

and context-awareness for MSN. Then, in Sections 3 we present CAMEO software architecture. Section 4 describes the main Android components used in the implementation and the description of the test application used to validate the prototype and its performances. The technical specification of CAMEO is presented in Section 5 with a detailed description of software packages and classes definition. Sections 6 and 7 describe the implementation of CAMEO MSN APIs, responsible for the management of the interactions between MSN applications and CAMEO. Finally, conclusions are presented in Section 8.

2 Context-awareness in opportunistic MSN

Currently the success of mobile applications and systems is directly tied to their potential impact on single individuals, groups of people sharing common interests and/or habits and, as a final goal, all the members of society. For this reason there are several application domains in which both research organizations and IT companies are investing by developing even more sophisticated mobile applications and systems, from personal health and family life to environment and social inclusion. Each of these areas contributes to improve the general well-being condition of people and their quality of life (see figure 2). The general paradigm of MSN can cover all these areas developing specific applications for each of them, but mobile systems' features can be further optimized by a general platform able to integrate and correlate context information derived from different application domains. For this reason CAMEO introduces a general notion of context defined as *well-being context*, which provide a two-fold awareness to the mobile system, *context- and social-awareness*, by correlating local information of both user and his device (like user's profile, interests, activities and local resources) with information derived from other users and devices, aimed at defining communication patterns among devices and social interactions among users. In this way, context information is used to optimize both lower-layer services, like networking and resources sharing, and upper-layer MSN applications.

It is worth noting that the ensemble of a user and his mobile devices represents the core entity of MSN to which we refer with the term "node". We define the context of a node as the integration of three main components: the *local* context, the *external* context and the *social* context (see Figure 3).

The *local* context represents all the information related to the local user and

Figure 2: Well-being application domains.

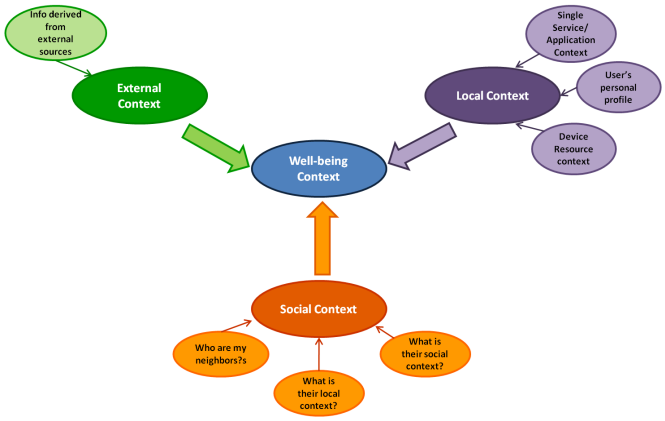


Figure 3: Well-being context.

his mobile devices including:

- the user's personal profile (i.e., all the information that describes interests and behavior of the user, like type and place of work, habits, life style, timetables). This information is generally provided directly by the user through dedicated interactions with the mobile device and its applications (e.g., through a digital agenda).
- the information collected by phone-embedded sensors (e.g., GPS, cameras, accelerometer, sound sensor), generally related to user's activity recognition and sensing of the external environment;
- information related to the mobile device resource management (e.g., battery level, available storage capacity, CPU occupancy, tasks management);
- context information specified by each application running on the mobile device (e.g., attributes of contents generated by specific MSN applications, like comments in Tourist-MSN or physiological parameters of Personal Health Systems, interests of users in specific contents).

The *external* context represents the set of information derived by external sources with respect to the user's mobile device. These sources are generally fixed or mobile stations aimed at monitoring specific parameters (e.g., air pollution, noise level, security cameras). This information can be integrated with participatory sensing data to improve the accuracy and fairness of detected information.

Finally, to better understand the definition of *social* context it is necessary to explain the reference scenario in terms of network of nodes. As previously cited, in MSN physical interactions among electronic devices within the same wireless transmission range (i.e., 1-hop wireless communications) reflect the physical proximity of users carrying the connected devices, creating thus (ephemeral) social *physical communities*. At the virtual layer, users are then grouped in *virtual communities* referring to their interests, habits or personal profiles. Due to users' mobility, physical communities can be characterized by a high dynamism, causing temporary and partial overlaps between physical and virtual communities.

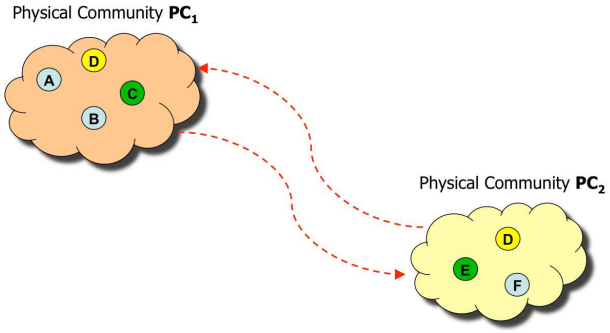


Figure 4: Example of physical communities with traveler nodes.

Connections among separate physical communities can be established through *traveler nodes* (i.e., nodes that while moving become members of multiple physical communities), which can be used as message ferries to disseminate content to interested users/devices that could never be directly connected with the source of that information. Figure 4 shows a simple example in which nodes belonging to the same physical community are characterized by different interests (highlighted with different colors), defining thus different virtual communities. Nodes moving between these physical communities can be used to carry data relevant to the different virtual communities they are in touch with.

The social context of a node represents the local view of the network obtained through 1-hop data exchange between nodes of the same social physical community. Specifically, by exchanging local context information among 1-hop neighbors, each node has a view of its current social context in terms of identification of its neighbors and their local context. In addition, since some nodes travel from a physical community to another, they collect social context information related to different physical communities. In our scenario, as it is generally assumed in the mobile social network literature [11], a *home* physical community is associated with each node, representing the community in which it spends more time and has stronger social links, while the traveling condition of a node is considered as a temporary visit to other communities driven by its social links.

The exchange of local context information allows also the local node to identify the virtual communities of its neighbors, grouping context information that identifies the same virtual community (e.g., tourists visiting Rome during Christmas holidays). In this way each node can maintain a map of encountered nodes belonging to different virtual and physical communities to develop optimized content dissemination protocols. To this aim, the social context of each node is composed of the social context related to its current physical community, and the historical social context related to nodes encountered in previously visited physical communities.

The aggregation and possible correlations of the three components of well-being context (local, external and social) allows the development of optimized MSN applications and services, making them able to identify the current situation the local node is immersed in and to take autonomous decisions for the

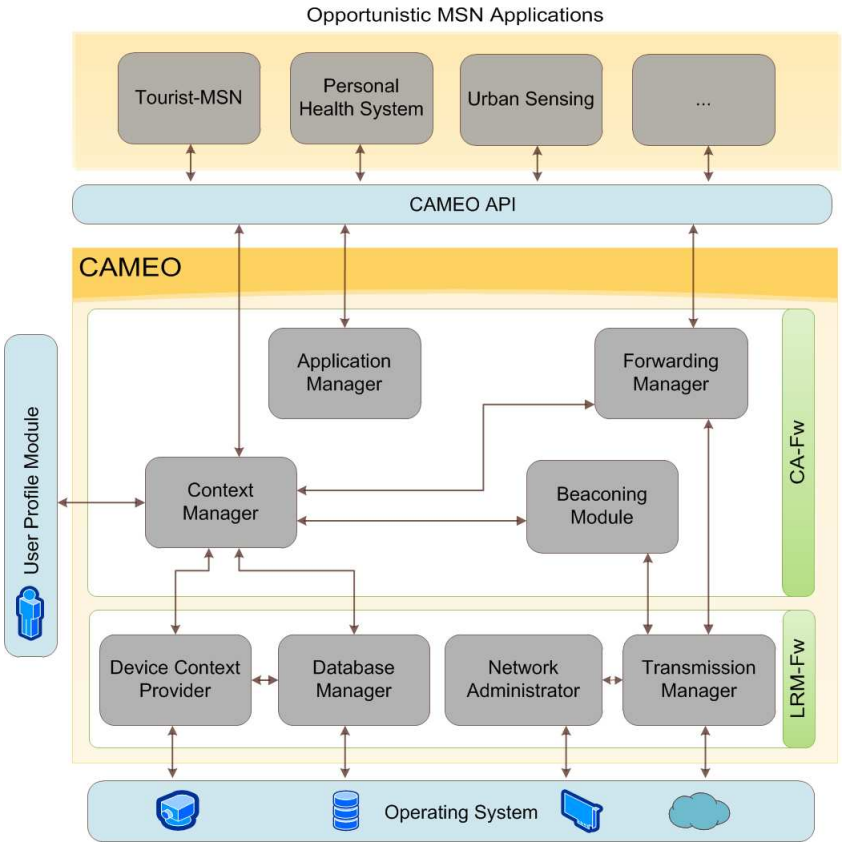


Figure 5: CAMEO software architecture.

entire system optimization.

3 CAMEO Software Architecture

CAMEO is designed as a light-weight and modular software architecture able to manage the multidimensional notion of well-being context.

As shown in Figure 5, CAMEO architecture is represented by a single software package containing two subpackages: **Local Resource Management Framework (LRM-Fw)**, aimed at implementing features strictly related to the interaction with the local resources of the device, both hardware (e.g., embedded sensors, capacity, battery) and software (e.g., communication primitives and programming libraries), and **Context-Aware Framework (CA-Fw)**, aimed at storing, elaborating and disseminating all the context information. In addition, CAMEO provides an API toward MSN applications and it directly interacts with an external module for user profile definition called User Profile Module.

3.1 Local Resource Management Framework

LRM-Fw is composed of three software modules:

Network Manager. In order to deploy real-time social networks, mobile devices must be able to exploit all the opportunities to communicate and exchange data through opportunistic communications. Therefore, the system must be able to interact with all the available wireless communication interfaces in order to decide the best communication medium under specific conditions. For this reason the Network Manager is responsible for the interaction of CAMEO with the wireless communication interfaces available on the mobile device (e.g., WiFi ad hoc, WiFi infrastructured, Bluetooth). It is also in charge of notifying other interested CAMEO components (specifically, the Transmission Manager) about the status of the connectivity between the local mobile device and his neighbors (e.g., wifi link status and quality information, Bluetooth active/not active status).

Transmission Manager. After the Network Manager has selected the wireless interface for the transmission of a specific message (both middleware or application messages), the Transmission Manager is in charge of establishing the communication channel between a source and a destination node through the use of standard communication primitives (e.g., socket, TCP/UDP protocols and related parameters). It also receives notification messages from the Network Manager in case of link errors or disconnection events toward the message destination node.

Database Manager. It is responsible for the interaction of LRM-Fw with the SQL database through operating system primitives. The database allows the permanent storage of selected context data to maintain the historical context profile and the definition of relationships among the context entities.

Device Context Provider. It is in charge of collecting context data derived from internal components of the mobile phone (e.g., embedded sensors, storage capacity level, battery level, resources consumption). Data specification and related parameters (e.g., sampling frequency for GPS or accelerometer, CPU occupancy threshold) are provided by the interaction of this module with the CA-Fw, following the directions provided by upper-layer applications or internal modules. Device Context Provider must be able to elaborate and combine data collected from multiple sources (either internal or external to the mobile device), generally characterized by proprietary data formats. In order to provide a uniform data format for sensors' readings, CAMEO integrates Device Context Provider functionalities with SensorML standard [13]. SensorML defines a standard model (called Observations & Measurements) to express data observations collected from sensors, enabling interoperability between different components. In addition, SensorML enriches the information related to observations with performance characteristics (like accuracy, and threshold values) and information related to sensors' discovery, location of sensor observations as well as processing of low-level sensor observations. SensorML is part of Sensor Web Enablement (SWE) framework [20], aimed at defining a standard model and encoding for the interoperability of sensor webs. Although SensorML is designed for the web, its extreme modularity makes it suitable also for mobile systems.

LRM-Fw directly interacts with operating system primitives and program-

ming libraries to access native functions like multi-thread and client-server communication protocols. For this reason, its implementation depends on the choice of the mobile device and its operating system. In CAMEO implementation we decided to use Google Nexus One smartphones running Android 2.2 due to the current success of this platform, but it can be easily adapted to other mobile platforms.

3.2 Context-Aware Framework

CA-Fw represents the core of CAMEO, being responsible for the collection and management of all the context information (local, external and social) and the development of internal context-based services (e.g., context aware forwarding protocols, context-based resource sharing services). It is composed of the following software modules:

Beaconing Module. It is responsible for the neighbor-discovery procedure inside the current physical community and the periodical exchange of context information among 1-hop neighbors. It allows each node to build up and maintain its social context. In order to avoid the periodically sending of large quantity of data, the Beaconing Module implements an optimized procedure based on a consistency check of Context Manager data structures. This procedure will be explained in detail after the description of Context Manager functionalities.

Forwarding Manager. It is responsible for the implementation of end-to-end communications. Specifically, it is designed to implement optimized forwarding protocols for opportunistic networks to successfully deliver a message to a multi-hop destination in case of intermittent connectivity (e.g., HiBOp forwarding protocol [10]).

Application Manager. It is in charge of establishing a communication channel between each MSN application and CAMEO through MSN API (see Section 4.1)

Context Manager. The main functionalities of Context Manager can be summarized in the following points: i) efficient storage and update of local, external and social components of well-being context; ii) elaboration of context information through the implementation of *context-based utility functions*. These features represent the core of context- and social-awareness implementation for both services and applications. Specifically, the result of an utility function can represent the utility degree of a content for a specific set of remote nodes, the ability of the local node to satisfy a remote request of resource sharing or the identification of the best next-hop neighbor to forward a message to a specific destination node. Currently, CAMEO prototype implements the utility function designed for content dissemination on opportunistic networks as detailed in Section 3.4.

For context storage and update, the Context Manager implements three separate data structures in order to provide efficient and reliable access methods to context information:

well-being Local Context table (wbLC): it contains the context information related to local context components divided in the following subtables: *User Profile* derived from ContextManager interactions with the User Profile

Module[2] (see Figure 5); *Application/Service Context* specified by each application developed on top of CAMEO and by single internal services; *Device Context* derived from the interaction of ContextManager with DeviceContextProvider; *External Context* as the set of information received from external sources, e.g. through sensing applications. This data structure encompasses the original notions of local and external context defined in Figure 3.

current community Social Context table (ccSC): it contains the list of current 1-hop neighbors of the local node and their context information disseminated through periodical beaconing messages.

historical Social Context table (hSC): it contains the list of communities previously visited by the local node associated with a timestamp as the temporal information of the last visit and a counter to maintain the number of visits in a predefined period of time. In addition, for each visited community hSC maintains the list of encountered nodes and their context. This information is essential to implement social-oriented policies (which will be described in more details in Section 3.4) for the evaluation of context-based utility functions. Due to its nature, this data structure needs non-volatile storage.

The content of ccSC represents thus a snapshot of the physical community of the local node in a specific instant. To collect this information, the beaconing procedure is in charge of efficiently exchanging relevant information of wbLC among the physical neighbors. Specifically, we assume that the Device context component of wbLC is not exchanged with other neighbors due to its real-time nature. In fact, Device context information is locally elaborated to evaluate the feasibility of specific actions (e.g., the local node receives a download request from a neighbor and it checks its local resources, like battery lifetime, before accepting and managing it), or represents information related to embedded sensors measurements, which generally has a limited temporal validity. Therefore, due to these strict time constraints, the validity of Device context is not compatible with the transmission over the network. Another component that does not necessarily require the periodical dissemination on the network is the External context. This information can be added to beaconing messages in case a service or an upper-layer application running on neighbors' nodes is interested in it (e.g., participatory and urban sensing applications). In this case it is necessary an explicit request by the application.

As far as the remaining wbLC components, they are disseminated by the Beaconing module with the following optimized procedure. Each node maintains a data structure for each context, representing the history of the changes made to that specific context. Each entry of this history is associated with a version number. Each beacon message contains the node identifier and a set of these version numbers, one for each context component. Every beacon interval T_b the Beaconing Module broadcasts a beacon message to 1-hop neighbors. Each neighbor checks in its data structures (ccSC and hSC) if it already has some information related to the sender node and, in case, if the stored context version corresponds to the version number embedded in the message. In case the neighbor has no context data related to the remote node, it directly sends a request

[2]We decided to design this module externally to CAMEO since it can be implemented as a stand-alone application dedicated to the collection of user's personal information independently of the running applications and services. However, the information provided by User Profile Module are integrated in wbLC with information provided by other services and applications that are mainly related to the user's interests, habits and so on.

message for the entire context. Otherwise, in case one of more version numbers do not correspond with those embedded in the message, it sends a request to the remote node specifying the version numbers it owns. Then, the remote node replies with the only context updates made after the received version number. In this way, the contexts are synchronized incrementally, reducing the amount of data periodically exchanged and reducing network overhead. Moreover, to ensure the scalability of the system, when the versions' history grows beyond a predefined limit, all the entries are collapsed into a unique version number, so that a limited amount of data is required to maintain an efficient context synchronization.

3.3 Community Detection

The dynamic identification of both current physical and virtual community of a node still represents an open research issue. In the last few years several algorithms for physical community detection have been presented in literature [16–18, 22] mainly based on the analysis of contact duration and number of contacts between pairs of users and their devices, assuming that individuals meet at a higher rate if they have one or more mutual friends. By correlating this information with the local context (and in particular with the personal user's profile), we should be able to define cause-effect relationships for which a specific user is in a specific physical community for a specific period of time (e.g., Bob is in the gym community because on Friday, 2pm he attends a yoga lesson). As far as the virtual community detection, it mainly depends on context modeling and reasoning. In fact, assuming that each virtual community is identified by a specific set of context information shared by involved users, the current virtual community of a node can be identified analyzing the correspondence of context values of the local node with those of nodes stored to its social context. This is relatively easy in case of exact match of the values, but finding general expressions for the elaboration and correlation of multidimensional information is a complex research issue that will not be analyzed in this paper.

In order to validate CAMEO in real testbeds, we refer to the simplified scenario explained in Section 2. In this case most nodes are static and only a few of them sporadically travel from a community to another. Therefore, virtual and physical communities of static nodes do not change and only traveler nodes dynamically change their physical community. We assume that each node declares in the User Profile a *home* community and each traveler node dynamically detects its current physical community by analysing the home communities declared by its neighbors. The community detection algorithm is formalized in the next paragraphs.

Considering N as the set of nodes of the entire network, and C as the set of physical communities declared by nodes of N, we can define the characteristic function $I_c(n, c)$ to indicate the membership of node $n \in N$ to physical community $c \in C$, where $c \subseteq N$ as:

$$I_c(n, c) = \begin{cases} 1 & \text{if } n \in c \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Assuming also that $Neigh_{n,t}$ defines the neighbor set of node n at time t and that $C_{neigh_{n,t}} \subseteq C$ is the set of communities declared by those neighbors at

time t, we can define the current physical community of node n at time t as

$$c_{cc}(n,t) = \underset{c_i \in C_{Neigh_{n,t}}}{\operatorname{argmax}} \sum_{n_j \in Neigh_{n,t}} I_c(n_j, c_i) \tag{2}$$

Therefore, every time a node receives a beacon from a new neighbor, CAMEO computes (2) through optimized accesses to CA-Fw data structures to discover the current physical community it is immersed in. This approach to community detection works well whenever most nodes are static or spend a limited amount of time outside their home community.

This is a simplified solution for community detection problem that allows us to validate the efficiency of CAMEO in detecting a change of the social context of the local node. More advanced solutions are under investigation [12]. The most effective will be implemented in CAMEO.

3.4 Context-based utility function for content dissemination

Generally, context-based utility functions are designed to evaluate context information in order to optimize specific features of services or applications. Since content dissemination represents a common functionality of MSN applications, the definition of a context-based function to determine the utility of a content for a set of users is essential to optimize content distribution over an opportunistic network. In fact, the selection of content to be disseminated strictly depends on users' interests and nodes' resource capabilities. In CAMEO we have exploited the utility-based content-dissemination framework proposed in [8]. Specifically, the utility function can be defined as follows:

$$U(c) = u_l(c) + \sum_{i \neq l} \omega_i u_i(c) \tag{3}$$

where $u_l(c)$ is the utility of a specific content for the local user, $u_i(c)$ is the utility for the i^{th} community the user is in contact with, and ω_i is a weight that defines the willingness of the user to cooperate with the i^{th} community (i.e., to spend its own resources to increase content availability for that community). In this way the local node offers its own resources to download contents available in its current physical community which can be useful for users belonging to previously encountered communities, assuming that in the next future the local node will visit again those communities.

The utility function u is defined by an application or an internal service and it is passed to the Context Manager through a dedicated interface by specifying the set of relevant context information and their evaluation algorithm. Every time a node changes its current physical community, the Context Manager uses the utility functions to evaluate the usefulness of the contents available in the new community for the node (or for the nodes it is in touch with).

A node can be characterized by different social behaviors by providing different weights to different communities. In [8] the authors have introduced and investigated a set of social-oriented policies describing 5 types of behaviors: i) *Uniform Social (US)*, in which all the visited communities assume the same weight; ii) *Present (P)*, which favours only the current community; iii) *Most Frequently Visited (MFV)*, evaluating the frequency of community changes; iv)

Future (F) and *Most Likely Next (MLN)* which require a probabilistic prediction of future visited community. Currently CAMEO supports both US and MFV policies, leaving to upper-layer applications the evaluation of the utility of a content for the current physical community. In fact, the applications often require a direct interaction with the final user to select an interesting content available on 1-hop neighbors. Therefore, as far as US, all the ω weights are set to 1 and the Context Manager calculates the utility function for all the previously encountered communities. For MFV, the ω weights are set proportionally to the number of visits of the local node to each community. In this way US policy facilitates data dissemination, because each node picks up all the contents found to be interesting for any previously encountered communities. On the other hand MVF policy reduces the data dissemination rate optimizing the local resources dedicated to preventive download procedure.

Other utility functions can be implemented by CAMEO for the optimization of specific services and applications as further explained in section 6.

4 Android implementation

To validate CAMEO functionalities and evaluate its performances, we implemented it on Android 2.2 platform. We have chosen Android due to its constantly rising popularity and because it naturally supports Java-based distributed and concurrent applications in addition to an easy access to system information like those related to embedded devices (such as GPS, sensors, camera). To better understand our implementation choices, we briefly introduce the definition of basic Android software components provided to developers. For additional technical details we refer the reader to [1].

The Application Framework is the main component of Android provided to the external developer. On top of this framework, developers can design their own applications with the same full access to APIs used by the core applications. An Android application is composed of four components: (i) *Activities*, representing the Graphical User Interface of the related application; (iii) *Services*, which allow the background execution of independent tasks; (iii) *Broadcast Receivers*, which listen for broadcast event communications among different applications and the other Android modules; (iv) *Content Providers*, which make a specific set of application's data available to other applications. The activation of the three first components and their following interactions are implemented through the *intent* mechanism: asynchronous messages exchanged between Application Framework components and containing the definition of the action to be performed.

Since CAMEO is a middleware platform supporting multiple concurrent applications, we decided to implement it as an Android Service. Thus, MSN applications, designed and developed to interact with CAMEO, are implemented as Android applications. A single instance of CAMEO, running on a separate process, is shared among all the applications. To support the communication between CAMEO and MSN applications, we exploit an Android technique based on the interprocess communication paradigm (IPC). Each application can define a set of special communication interfaces visible to the other applications by which they can exchange data. The interfaces are defined through a proprietary interface definition language called AIDL (Android Interface Definition

Language) similar to other popular languages based on CORBA or COM specifications (e.g., Java CORBA IDL [5] and C++ CORBA IDL [2]). The data that can be transferred through AIDL interfaces is limited to Parcelable objects, which are designed as a high-performance IPC transport objects. This mechanism allows fast data exchange to the detriment of limited design flexibility due to the lack of standard functions for the marshalling of Parcelable objects.

As far as the support for direct connection between devices, Android does not allow the activation of Wi-Fi ad hoc mode on its devices either through the standard graphical interface or programming libraries[3]. Nevertheless, due to the open source nature of Android, it is quite simple to overcome this limitation. Specifically, in the last few years a lot of customized aftermarket firmwares based on Android Open Source Project (AOSP) were released by third parties. Some of these natively supports ad-hoc mode, avoiding the developer to be bound to the tedious work of building his own ad-hoc ready firmware. In order to have a stable development platform and to allow fast 1-hop opportunistic communications, we decided to install CyanogenMod 6.1.0-N1 firmware [3] on our smartphones. CyanogenMod 6 is based on Android 2.2 AOSP code and adds a lot of useful functionalities in addition to the ad-hoc mode support (see [3] for details). The devices equipped with the last version of Android (4.0) and some phones running Android 2.3 with proprietary operating system extensions support Wi-Fi Direct standard. This standard allows terminals to communicate directly with each other using Wi-Fi connectivity, in a way similar to ad hoc mode. Some research work has been done to support opportunistic networking over Wi-Fi Direct [21]. In future work we will assess the feasibility of adopting Wi-Fi Direct as the communication standard for CAMEO-enabled devices. At present, the pure Wi-Fi ad hoc mode provides more flexibility and a full support for opportunistic communication paradigms.

4.1 CAMEO APIs toward MSN applications

CAMEO defines two different APIs to provide social- and context-aware functionalities to MSN applications. The first one manages MSN applications requests toward CAMEO, while the second one handles CAMEO notifications toward the applications (e.g., messages, events, errors). In the following we summarize the features provided by the two APIs; the complete specification can be found in Sections 6 and 7.

- **Registration**. Each application must be registered to CAMEO in order to obtain the access to all the internal functionalities. During the registration a unique identifier is assigned to the application.
- **Application Context specification**. Each application specifies to CAMEO the set of context information relevant for its execution. This information is stored in wbLC table and disseminated over the network through the beaconing procedure as part of the local context of the node.
- **Utility function evaluation**. MSN applications can ask CAMEO to implement the algorithm for utility function evaluation. To this aim the

[3]This is still valid for the current version including the 2.2 stock firmware shipped with Google Nexus One

application must provide to CAMEO the algorithm and related parameters.

- **Message sending/receiving**. Applications can send/receive messages toward remote nodes. They are notified in case of failure during a message sending. Since Android launches an independent instance of Dalvik JVM [4] for each process with a predefined memory size (32MB), the system can present overload problems in case of exchange of big application contents. To avoid this situation, CAMEO implements a file segmentation procedure splitting the requested content into fixed length data chunks (512Kb each). The correct reception of each chunk is acknowledged, so that the sender node can manage automatic retransmissions of not acknowledged chunks.

CAMEO notifications toward MSN applications are implemented using callback interfaces created during the registration procedure. To manage different concurrent applications, CAMEO maintains the list of registered and currently active callback interfaces assigning a logical communication port to each of them. A special communication port is used by the Context Manager for context data exchange over the network and its interaction with the other internal modules of CAMEO. CAMEO notifications are related to the following events:

- **Application content discovery**. Every time CAMEO finds a new application content from a remote node it informs the interested application.
- **Neighbors discovery**. CAMEO informs the applications when a neighbor enters/exits the 1-hop area.
- **Community discovery**. CAMEO informs the applications when the current community changes.

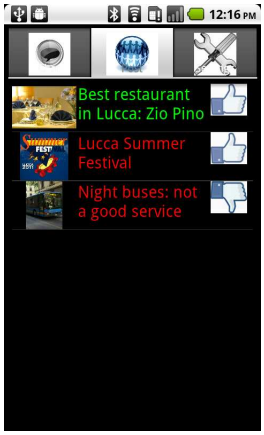
4.2 Tourist-MSN Application

Tourist-MSN [7] is a real example of MSN application developed on top of CAMEO. It is aimed at improving people experience during tourist visits by allowing individuals to create, collect and share useful information related to geolocated points of interest (POIs) through opportunistic wireless communications among their mobile devices. Tourists can create multimedia contents denoted as posts. They are characterized by a title, a textual content (to comment or express impressions related to the POI), and optional information like audio files, images or videos. Posts are divided into categories (e.g., events, cultural visits, transportation) in which users can express their interests. Tourist-MSN implements also an opportunistic text chat that allows a limited group of users to communicate in a quasi real-time fashion. Each chat is identified by a title and a category.

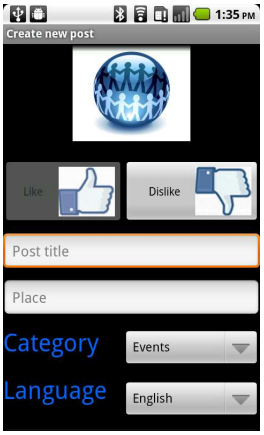
TouristMSN specifies through MSN API the following information as Application Context to be disseminated over the network by CAMEO: (i) title and category for each post and chat generated by the local user; (ii) user's interests in specific categories of posts and chats.

Therefore, through context dissemination each node becomes aware of the available contents in its current physical community. However, due to intermittent connectivity conditions characterizing opportunistic networks, the manage-

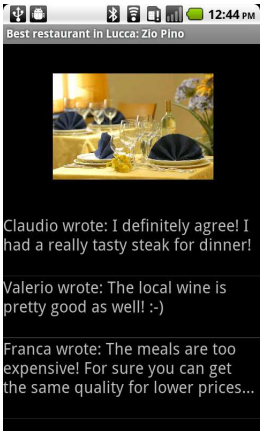
ment of a real-time chat can be developed only among 1-hop neighbors. Instead posts' distribution results in an asynchronous content exchange. Users can increase the content of a post/chat by adding their own comments. Subsequently CAMEO distributes the updates on the network.



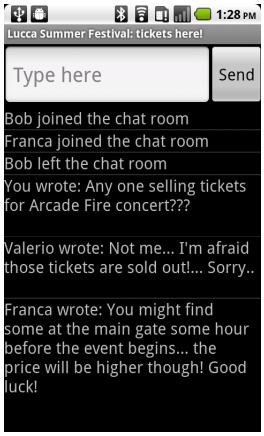
(a) List of posts. In green: posts already downloaded. In red: posts available for download



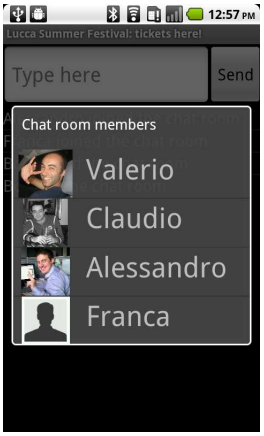
(b) Interface for the creation of new posts



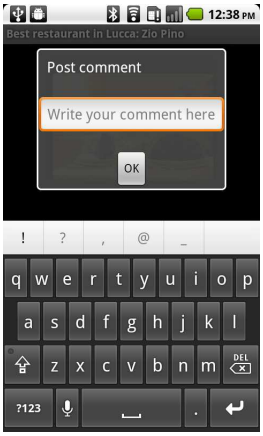
(c) Post comments



(d) Realtime opportunistic chat



(e) Chat members



(f) Add a new comment to an existing post

Figure 6: Screenshots of Tourist-MSN Graphical User Interface

Every time a new post or a new chat matching the interests expressed by the user becomes available from a neighbor node, CAMEO notifies the application with an event message. The application then informs the user through the GUI about the availability of the new content and the user can decide whether to download the post (or join the chat room) or not. At the same time, CAMEO evaluates the utility function for contents available in the current physical community with respect to the interests of previously encountered users by implementing US and MFV social-oriented policies described in Section 3.4. Then,

the subset of contents that maximizes the utility satisfying local nodes's requirements (e.g., memory availability, permanence time in the current community) are downloaded.

Figure 6 shows some screenshots of Tourist-MSN application running on Google Nexus One.

5 CAMEO Technical Specification

In this Section we give a detailed description of CAMEO software architecture, starting from software packages up to class definition. All the classes have been written in Java programming language for Android operating system.

CAMEO is composed of the following software packages:

- cnr.CAMEO
- cnr.CAMEO.CAFw
- cnr.CAMEO.CAFw.ContextManaging
- cnr.CAMEO.LRMFw
- cnr.Common

5.1 cnr.CAMEO

This package contains basic classes of CAMEO, such as the class of the main GUI launched when CAMEO starts running, some custom exceptions classes and the external module dedicated to user context management (User Profile Module). The package is composed of the following classes:

- `MainActivity`: extends `Activity` (Android). This class represents the main GUI of CAMEO and is the first class launched by CAMEO. The purpose of the `MainActivity` class is to provide a simple graphical interface by which the user can control the execution of CAMEO service (i.e., start and stop its execution). Moreover, the GUI contains a button by which the user can launch the `UserProfileModule` `Activity`.
- `RemoteListEntry`: This class defines an entry of the Android `Callback-RemoteList`, the structure where the applications callback interfaces are maintained. CAMEO uses these callback interfaces to communicate with the registered applications. When an application registers to CAMEO, a `RemoteListEntry` is added to the `RemoteCallbackList`. The entry is removed in case of application log out or crash.
- `ServiceManager`: extends `Service` (Android). It is the main component of CAMEO as the Android `Service` that, running in background, provides all context-aware and opportunistic features to the registered applications. The `ServiceManager` is launched by the `MainActivity` and is responsible for the creation of all the components of CAMEO CA-Fw (i.e., the `ContextManager`, the `Forwarding` module, and the `Beaconing` module). The functionalities of the `ApplicationManager` (see Section 3.2) are implemented inside the `ServiceManager`, indeed the `ServiceManager` implements the AIDL interface `PlatformInterface`, which provides CAMEO

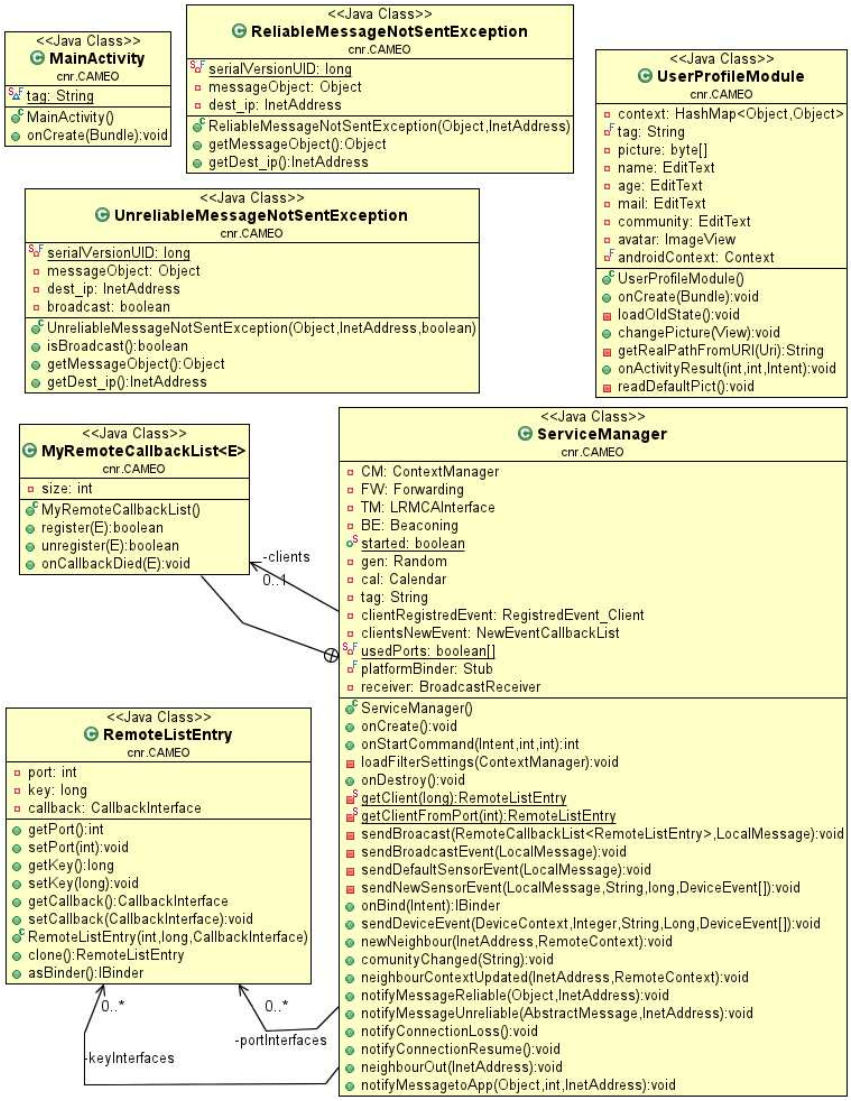


Figure 7: cnr.CAMEO package - UML class diagram

MSN APIs to the applications. The `ServiceManager` takes also care of the following tasks:

1. to register each application that wants to interact with CAMEO, setting a unique identifier (i.e., a port number) for it;
2. to receive and process requests coming from the registered applications (see Section 6 for further details);
3. to notify the applications when an event occurs or a message is received (using the identifier to select the right application). A detailed description of this notifications can be found in Section 7.

- `UserProfileModule`: extends `Activity` (Android). This class defines the module in charge of acquiring and managing the user profile (e.g., preferences, personal profile, ...). The `UserProfileModule` is a GUI component that allows the user to insert and successively modify his personal information into a set of pre-defined fields (e.g., name, gender, age, home community, preferences).

- `ReliableMessageNotSentException`: extends `Exception`. This class defines a custom exception used to handle the errors generated when CAMEO tries to send a message using the reliable transmission protocol(TCP), but the `TransmissionManager` (package cnr.CAMEO.LRMFw) has not been started yet or it is in idle state.

- `UnreliableMessageNotSentException`: extends `Exception`. This class defines a custom exception used to handle errors generated when CAMEO tries to send a message using unreliable transmission (UDP), but the `TransmissionManager` (package cnr.CAMEO.LRMFw) has not been started yet or it is in idle state.

5.2 cnr.CAMEO.CAFw

This package contains all the classes related to the Context-Aware Framework (CA-Fw, see Section 3.2 for further details on the services provided by this framework). The package is composed of the following classes:

- `Beaconing`. This class represents the `BeaconingModule` in charge of generating periodic messages, also known as beacons (defined by the `BeaconMessage` class), used by CAMEO to advertise the presence of a node towards the 1-hop neighbors in the network. `Beaconing` uses a Java `Timer` to generate the beacons periodically and passes them to the `TransmissionManager`, which is responsible for sending the messages over the network. Beacons are also used to disseminate the version numbers of the nodes' contexts, informing the remote nodes of the presence of available contents.

- `BeaconMessage`: extends `AbstractMessage`. This class defines the structure of a beacon message. It contains a `BeaconContent` and a timestamp that reflects the beacon creation time. The timestamp is used in conjunction with the contexts' version numbers. If CAMEO crashes or is restarted, the version numbers are reset to zero. The timestamp is thus used to identify contexts with the same version number, but with possibly different content.

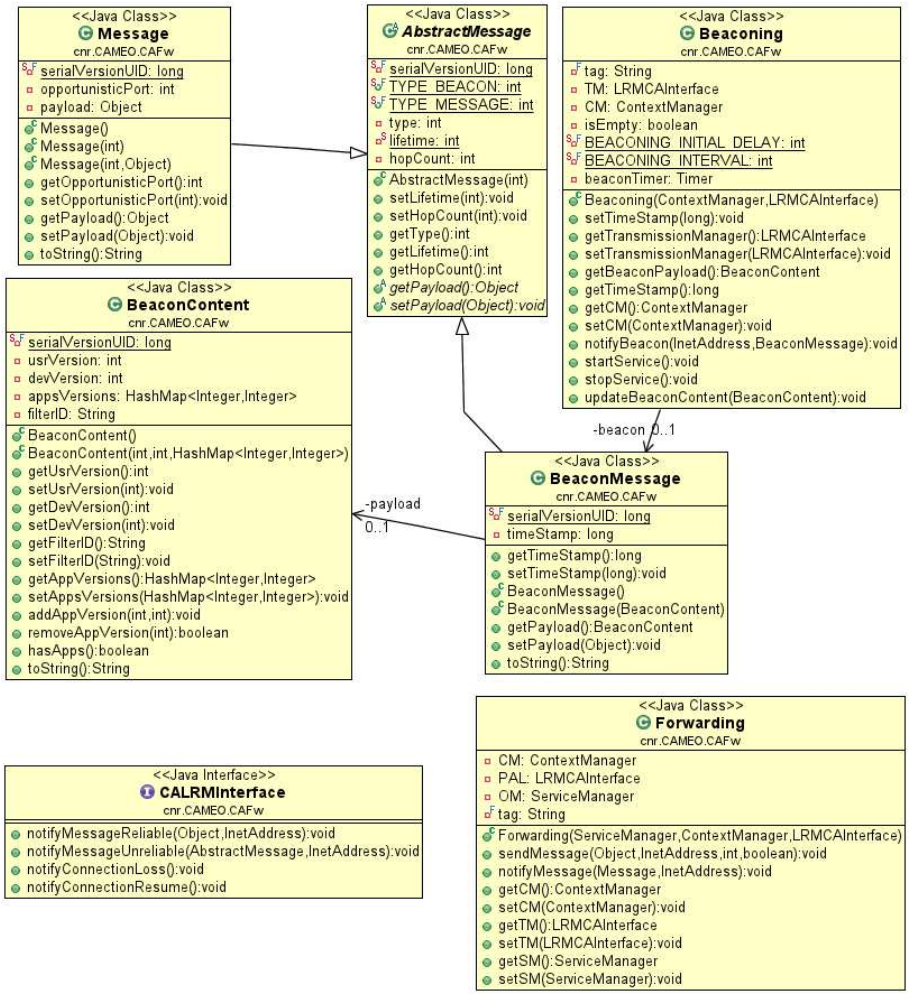


Figure 8: cnr.CAMEO.CAFw package - UML class diagram

- `BeaconContent`: implements `Serializable`. This class defines the content of a beacon message. This content is formed of a set of version numbers related to the contexts of the user and of the applications running on CAMEO which a node wants to send over the network. The `BeaconContent` also contains the ip address of the sender, which is used as unique identifier for the nodes.
- `CALRMInterface`. This interface is used for the communication between the Local Resource Management Framework (LRMFw) and the Context-Aware Framework (CAFw). The modules of the LRMFw can send their notifications toward the CAFw using this interface, without knowing which module will elaborate the requests, ensuring the modularity of the system. It is implemented by `ServiceManager`.
- `Forwarding`. This class is designed to implement an opportunistic routing algorithm. Currently the class is empty and CAMEO uses only 1-hop communication. In future work we plan to implement a context-based forwarding protocol as HiBOp [10].
- `AbstractMessage`: implements `Serializable`. This is an abstract superclass defining a generic message, containing the properties and methods in common between the `BeaconMessage` class and the `Message` class.
- `Message`: extends `AbstractMessage`. This class defines a message to be exchanged over the network between applications through CAMEO. The message has a payload field, inherited from the superclass `AbstractMessage`. This generic payload represents the content of the message. To deliver the message to the right application at the destination node, CAMEO assign a port number to each registered application and each `Message` is marked with the respective port number.

5.3 cnr.CAMEO.CAFW.ContextManaging

This is a sub-package of `cnr.CAMEO.CAFW` containing all the classes related to the management of context information. The content of the package is the following:

- `AbstractContext`: implements Serializable. This is a superclass representing a generic context. The `AbstractContext` contains a version number and an `HashMap` representing the key-value pairs related to a context.
- `ContextManager`: implements `DeviceContextUpdater`. This is the main class for the management of context data. The `ContextManager` is responsible for the maintenance of context information related to the user, the device and the applications involving the local node and the remote nodes encountered in the network. It is in charge of exchanging `ContextMessages` between nodes in the network, performing the synchronization of context data between peers. The `ContextManager` manages the `ContextTable`, adding, updating or removing context information related to remote nodes when needed. Moreover, it maintains the `LocalContext`, containing the context data of the local node. Through the `DeviceContextUpdater` the `ContextManager` exposes the methods used by the applications to modify the local device context. The `ContextManager` checks

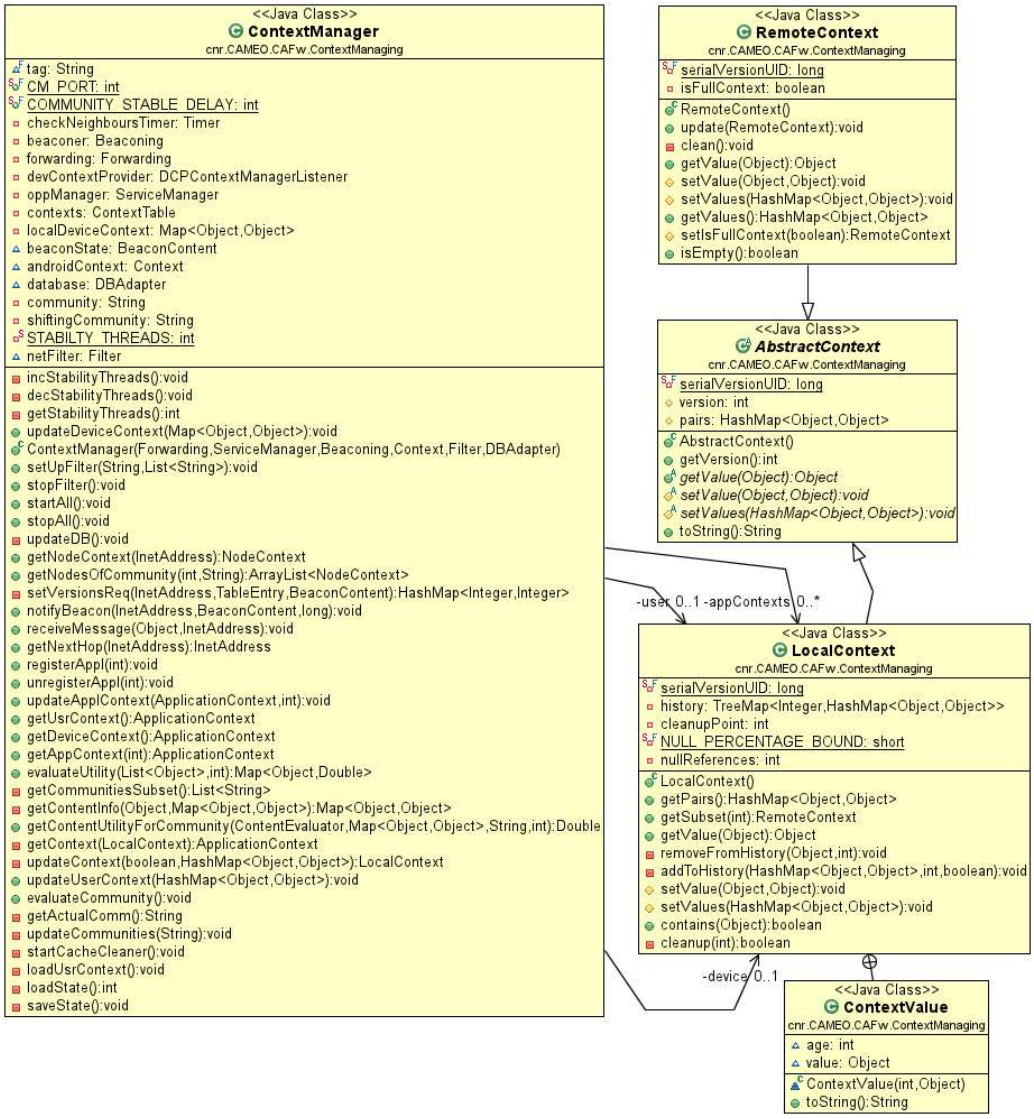


Figure 9: cnr.CAMEO.CAFw.ContextManaging package - UML class diagram (1)

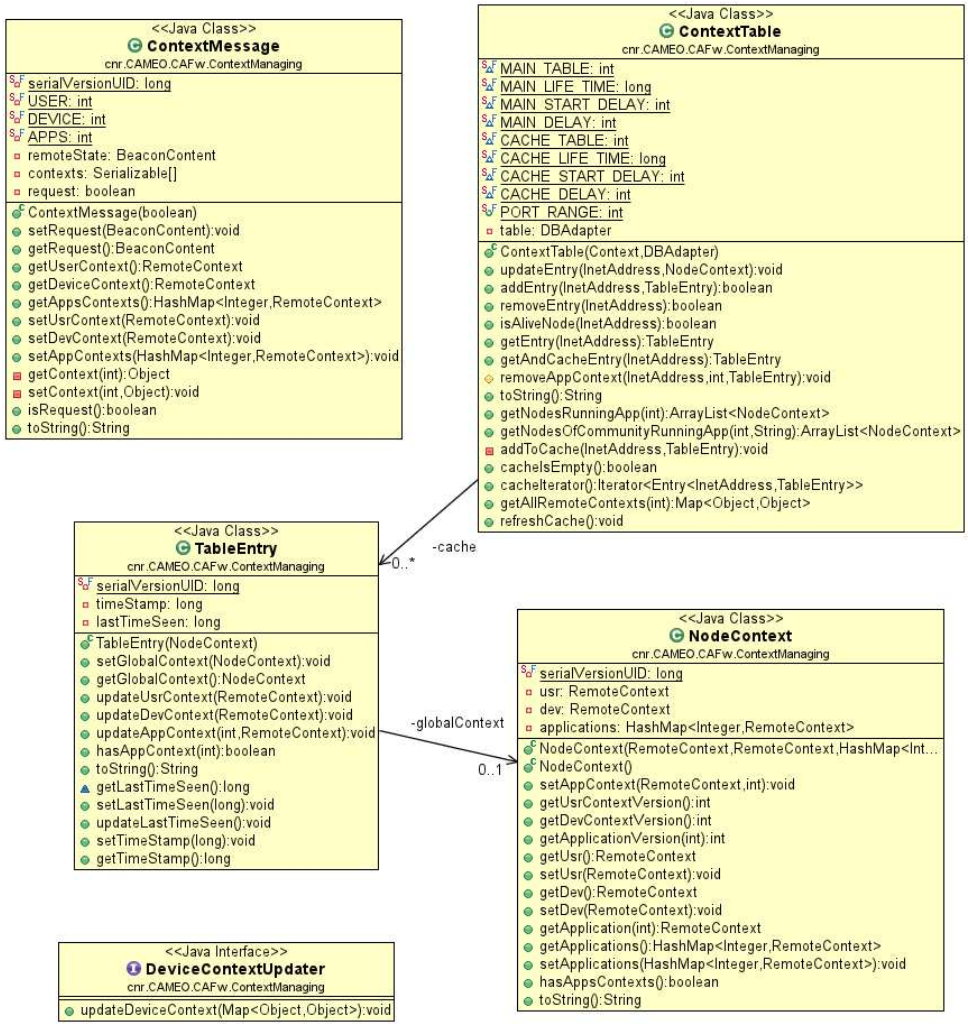


Figure 10: cnr.CAMEO.CAFw.ContextManaging package - UML class diagram (2)

the beacons received from the `Beaconing` module to find if a neighbor has entered or quit the 1-hop area. When a new neighbor is detected, the `ContextManager` adds an entry to the `ContextTable`. Then, it periodically checks the ccSC table (see Section 3.2 for further details) to determine if a neighbor has left the 1-hop area. The `ContextManager` also evaluates, after an explicit request received from an application, the utility of one or more application content for one or more nodes of a community or a set of communities using the utility function provided by the application and adopting the social-oriented polices required by the application (see Section 3.4). The `ContextManager` is also responsible for the discovery of the current community of the local node.

- `ContextMessage`: implements `Serializable`. This class defines a special type of message used to exchange context information between nodes in the network. The `Beaconing` module advertises available context data inside the 1-hop neighborhood. If a node finds new available context information from a new neighbor, or a new version of a context from a known neighbor, it sends a request to the remote node through a `ContextMessage`, eventually indicating the version of the context it already has to perform an incremental synchronization. The node which receives a context request sends its whole context (or part of it with respect to the version number received from the other node) to the requester, using a `ContextMessage`. The `ContextMessage` is used both for user contexts and application contexts.

- `ContextTable`. This class represents the data structure holding the context information related to remote nodes. Specifically, this class implements both the data structures defined as ccSC and hSC (see Section 3.2). The ccSC table is implemented as a Java `Map` object, where the ip address of the remote nodes is used as index of the table and each entry contains the context data of the remote node. When a new neighbor is detected, the `ContextManager` notifies the `ContextTable`, which adds an entry to the ccSC table. If the beacons of a neighbor are not received for a certain period of time, the `ContextManager` asks the `ContextTable` to remove the respective entry from the ccSC table. In this case the `ContextTable` moves the information related to the node that has left the 1-hop area from the ccSC table to the hSC table. The hSC table is implemented as a SQLite database and accessed by CAMEO using the SQLite APIs offered by Android. The ccSC and the hSC are also accessed by the `ContextManager` during the evaluation of the utility function for a given application content.

- `LocalContext`: extends `AbstractContext`. The `LocalContext` represents the context information related to the local node, including the application contexts and the user context. It is the implementation of the well-being Local Context table (wbLC) (see Section 3.2 for further details). This data structure is implemented as a set of Java `HashTable`, each of which represents a context (i.e., user, device, external and application) and contains its key-value pairs. The information of the `LocalContext` is sent to the other nodes in the network using `BeaconMessage` and `ContextMessage` objects. When the `ContextManager` receives a `ContextMessage` with a

request for a certain context with a specified version number, the `LocalContext` checks if the version number and the timestamp provided by the remote node are valid, then it returns the subset of information required by the remote node to synchronize its context data with the local data. Each change related to the data of a certain local context is tracked within a Java `TreeMap` object called `history`, so that each entry of the `history`, indexed by a version number, points toward the key-value pairs modified during the update. Each application can provide multiple updates at the same time to avoid frequent context updates which can lead to network congestion due to the potentially large amount of data to be exchanged for the synchronization procedure. Subsequent updates of the same context key-value pairs are collapsed into the last version number of the `history` data structure to reduce the dimension of the `TreeMap`. If the number of updates of a context data exceeds a pre-defined threshold, all the versions are collapsed into the last version number of the `history`.

- `NodeContext`: implements `Serializable`. The `NodeContext` class defines a wrapper for all the context information related to a remote node and represents a generic entry for the hSC of the `ContextTable` object. The `NodeContext` object contains the user context and the list of application contexts related to a remote node.

- `RemoteContext`: extends `AbstractContext`. This class defines a generic context related to a remote node. A `Remote context` contains the key-value pairs that defines the context.

- `TableEntry`: implements `Serializable`. This class represents an entry of the ccSC table, indexed inside the `ContextTable` with the ip address of the respective remote node. Each `TableEntry` contains a `NodeContext` object and a timestamp, used to discover a `neighbor_out` event (in case the beacons of a certain remote node are not received for a pre-defined period of time).

- `DeviceContextUpdater`. This interface allows the ContextManager to update the status of the local device context following the input of Local Resource Management Framework .

5.4 cnr.CAMEO.LRMFw

This package contains all the classes concerning the Local Resource Management Framework of CAMEO. This framework is in charge of performing all the lower level tasks, such as the exchange of messages over the network, the management of the wireless network interface, the management of the SQLite database and the access to the information related to the hardware sensors of the mobile device. The package is composed of the following classes:

- `ContextProvider`: extends `Thread`, implements `DCPContextManagerListener`. The `ContextProvider` is in charge of acquiring information regarding the context of the device, including sensors data, battery status, memory usage, etc. The `ContextProvider` periodically checks the status

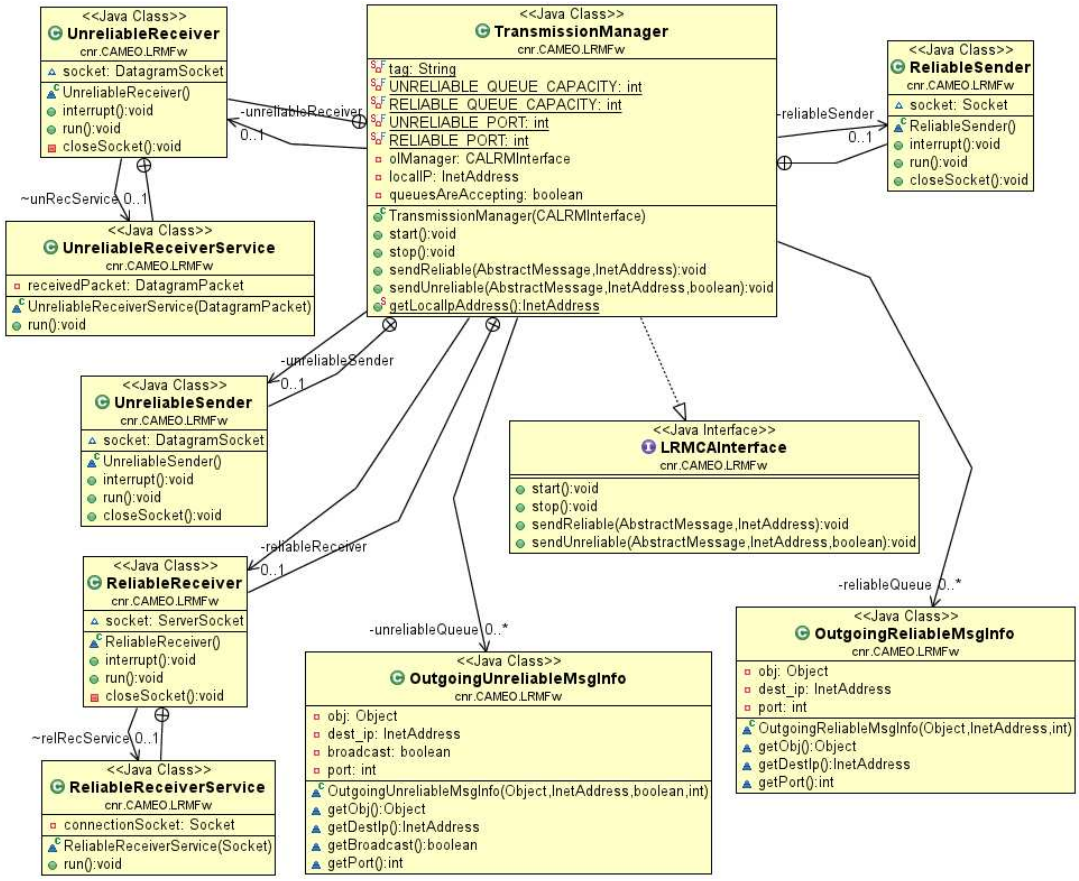


Figure 11: cnr.CAMEO.LRMFw package - UML class diagram (1)

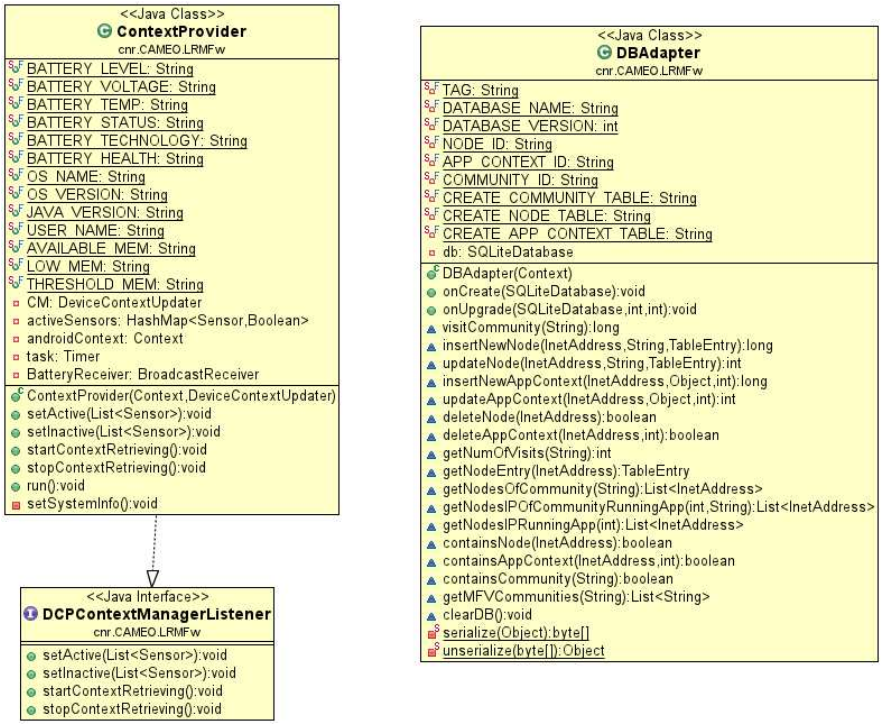


Figure 12: cnr.CAMEO.LRMFw package - UML class diagram (2)

of different hardware components with different sample rates. These parameters are defined by the `ContextManager` class of the CA-Fw based also on applications' requirements.

- `DBAdapter`: extends `SQLiteOpenHelper` (Android). This class contains all the methods used to access and manage the SQLite databases.
- `DCPContextManagerListener`. This interface allows the `ContextManager` to retrieve information concerning the device, gathered through the `ContextProvider`.
- `LRMCAInterface`. This interface defines the methods to be used by the CAFw to communicate with the LRMFw.
- `NetworkManager`. It interacts with Android to manage the wireless network interface. It can request Android to activate/deactivate the network interface and it listens for status changes of the interface.
- `OutgoingReliableMsgInfo`. This class represents a generic message stored inside the outgoing queue of the `TransmissionManager`. Each of these messages has a destination address, a port (used by CAMEO to identify the destination application) and a payload containing the message itself. This class is related to a message to be sent using the reliable communication protocol (TCP).

- `OutgoingUnreliableMsgInfo`. This class represents a generic message stored inside the outgoing queue of the `TransmissionManager`. Compared with the `OutgoingReliableMsgInfo`, this class concerns unreliable communications (UDP).
- `TransmissionManager`: implements `LRMCAInterface`. The `TransmissionManager` is responsible for the data transmission over the network. It provides two different types of communication: reliable and unreliable. The former uses Java `ServerSocket` and `Socket` objects, while the latter uses `DatagramSocket` objects for data exchange. The `TransmissionManager` maintains two messages queues (for reliable and unreliable communication respectively).

5.5 cnr.CAMEO.LRMFw.Sensors

This package contains all the classes aimed at managing the collection of data coming from the different hardware sensors. The names of the classes are self-explanatory. The content of the package is the following:

- `AccelerometerManager`
- `GyroscopeManager`
- `LightManager`
- `MagneticFieldManager`
- `OrientationManager`
- `ProximityManager`
- `PressureManager`
- `TemperatureManager`

5.6 cnr.CAMEO.common

This package contains the classes shared between CAMEO and MSN applications, which must be imported by applications as external libraries to allow the application's access to CAMEO functionalities.

- `ApplicationContext`: implements `Serializable`. This class represents the context related to a specific application as key-value pairs directly specified by the application. Each application can store or retrieve its own `ApplicationContext` object using CAMEO MSN API.
- `ContentEvaluator`: implements `Serializable`. This class is aimed at giving a simple way to define a set of criteria to be used by the `ContextManager` during the evaluation of the utility function. Each application can create its own `ContentEvaluator`, defining a list of fields (called evaluators) which represents the tags of the application contents to be matched with the respective preferences of the user. The application can weight the different evaluators with a value between 0 and 1. The utility function is computed as the weighted sum of the binary results of the matches of

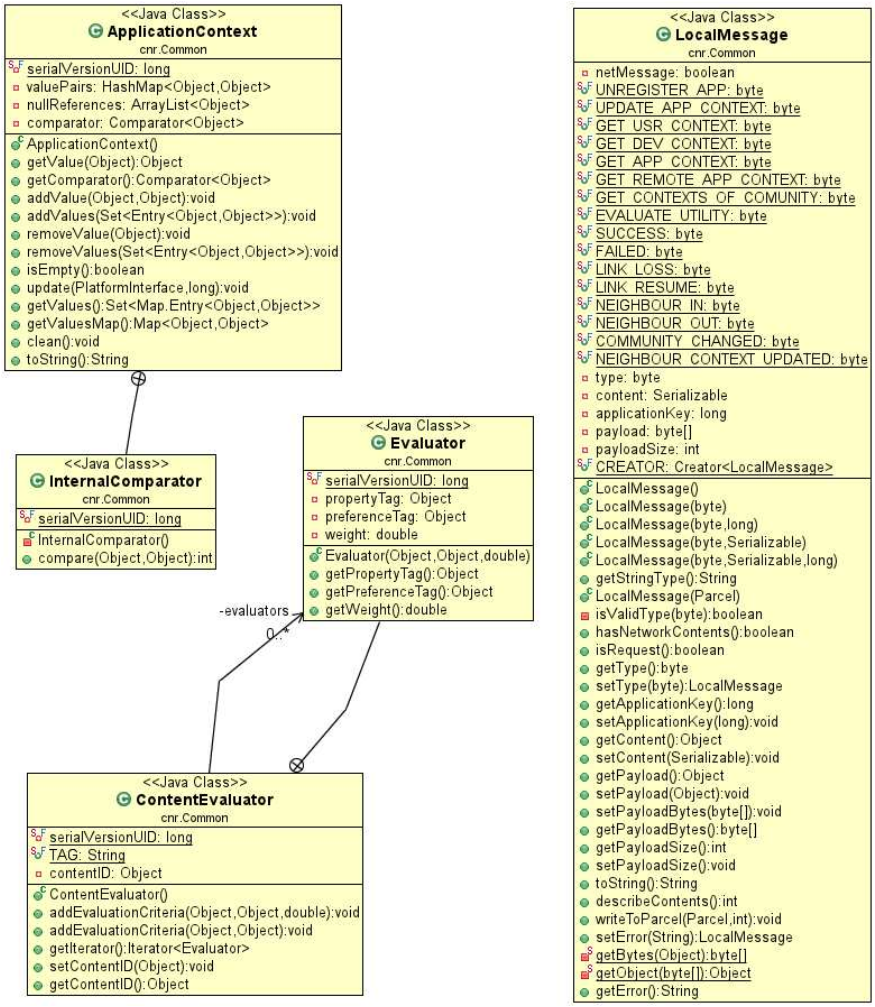


Figure 13: cnr.CAMEO.Common package - UML class diagram

the fields defined by the evaluators. For example, an hypothetical application exchanging music content over the network defines two evaluators, named "genre" and "format" respectively, the former mapping the musical genre of an audio file with the preference of the user regarding musical genres and the latter identifying a link between the file format of the application content (e.g., mp3, wav, ...) and the preferred format of the user. The `ContextManager` matches the values mapped by the evaluators, returning 1 if the value related to the content and the preference of the user match, and 0 otherwise. The utility function calculated by the `ContextManager` for this application would be the sum of the values returned by the match of the two evaluators multiplied by their respective evaluator weights, then multiplied by the weight given to the community to which the involved nodes belong to, defined by the chosen social-oriented policy (see Section 3.4).

- `LocalMessage`: implements `Parcelable` (Android). This class defines the format of the messages exchanged between the applications and CAMEO. A `LocalMessage` can be an application request sent by the applications toward CAMEO or a CAMEO event sent toward the applications. A `LocalMessage` contains a `TYPE` field, which determines the type of request or notification. It contains also additional information (e.g., an application message to be sent over the network or additional parameters required by some requests) placed in two different optional fields. The first field is called `content` and is used for local communication, while the second field is called `payload` and it is used for remote communication. The different types of `LocalMessage` are explained in more details in Section 6.

In addition, CAMEO provides two AIDL interfaces to manage the communication between the middleware platform and each MSN application. Specifically:

- `MSNInterface` (AIDL). This interface contains the methods offered by CAMEO to the applications. The application directly uses MSNInterface object to call CAMEO methods.
- `CallbackInterface` (AIDL). This interface defines the methods called by CAMEO to send various notifications to the applications. The application must implement this interface and pass the instantiated object to CAMEO, which uses it to send each notification to the application.

  These interfaces are detailed in the following Sections.

6 MSN Interface

MNS interface provides the following functionalities to MSN applications.

6.1 Register with CAMEO

- **Name of the service primitive:** `RegisterClient`
- **Description:** This method allows an application to register with CAMEO. CAMEO approves the registration if the port specified by the

application is not yet in use, otherwise it denies the registration. If the registration is approved, CAMEO adds the interface provided by the application during the registration to the callback interface list and uses it to communicate back to the application. A registered application can access the services provided by CAMEO, using all the other methods described in this Section. If an application is not registered, it can only use the `RegisterClient` primitive. A call to any other method will cause CAMEO to reply with an error message.

- **Type:** Confirmed, Synchronous, Local
- **Semantics:** The primitive shall provide the following parameters:
  1. *Port* (input)
     - Type: integer
     - Accepted values: a valid integer
     - Description: The port number used as unique `ID` to redirect the notifications and the messages received by CAMEO over the network to the right application.
  2. *Callback* (input)
     - Type: `CallbackInterface`
     - Accepted values: a valid `CallbackInterface` object
     - Description: This is the interface that the application must provide to CAMEO in order to register. CAMEO uses these interfaces to communicate back with the applications.
  3. *LocalMessage* (output)
     - Description: This is the returned value containing the result of the registration procedure.
     - Returned value: `FAILED` if the registration procedure has failed or `SUCCESS` otherwise.
- **When generated:** This primitive is generated when an application wants to register with CAMEO.
- **Effects of receipt:** The receipt of this primitive by the `ServiceManager` (which implements the `PlatformInterface` interface) causes CAMEO to start the registration process. The `ServiceManager` checks if the port provided by the application is already in use. If this is the case it denies the registration, otherwise it accepts it.

6.2 Send a Generic Local Request

- **Name of the service primitive:** `SendLocalRequest`
- **Description:** This is a generic primitive provided by CAMEO to allow the applications to request various types of services.
- **Type:** Confirmed, Local
- **Semantics:** The primitive shall provide the following parameters:

1. *request* (input)
   - Type: `LocalMessage`
   - Accepted values: a valid `LocalMessage` object
   - Description: This parameter contains the specification of the request that the application wants to send to CAMEO. The request type is specified inside the field `TYPE` of the `LocalMessage` object. Additional content required by some requests can be placed inside the field `Content` of the `LocalMessage` object.
2. *LocalMessage* (output)
   - Description: This is the returned value containing the result of the request.
   - Returned value: `FAILED` if the request has failed or `SUCCESS` otherwise.

- **When generated:** This primitive is generated when an application wants to request a service to CAMEO.
- **Effects of receipt:** The receipt of this primitive by the `ServiceManager` (which implements the `PlatformInterface` interface) causes CAMEO to evaluate and perform the actions required by the application, depending on the type of request the applicationhas made. The different types of requests that the application can send to CAMEO are listed below:

6.2.1 Unregister with CAMEO

- **TYPE of LocalMessage:** `UNREGISTER_APP`
- **Description:** This type of local request is used by the applications that want to cancel their registration with CAMEO.
- **Additional parameters:** none
- **When generated:** This local request is generated when an application wants to unregister with CAMEO.
- **Effects of receipt:** The receipt of this primitive by the `ServiceManager` causes CAMEO to unregister the application and return a `LocalMessage` object with the `TYPE` field set to `SUCCESS`.

6.2.2 Update Application Context

- **TYPE of LocalMessage:** `UPDATE_APP_CONTEXT`
- **Description:** This type of local request is used by the applications that want to update their application context (i.e., add, modify or delete the key-value pairs of the context data).
- **Additional parameters:** The application context that the application wants to store instead of the original one, saved into an `ApplicationContext` object.
- **When generated:** This local request is generated when an application wants to update its application context stored inside CAMEO.

- **Effects of receipt:** The receipt of this primitive by the `ServiceManager` causes CAMEO to update the context information related to the application that called the primitive. CAMEO distributes the new application context over the network, adding a new version number and performing the operations described in Section 5.3 regarding the `LocalContext` class. The request always returns a `LocalMessage` with the field `TYPE` set to `SUCCESS`

6.2.3 Get User Context

- **TYPE of LocalMessage:** `GET_USR_CONTEXT`
- **Description:** This type of local request is used by the applications that want to retrieve the user context information.
- **Additional parameters:** none
- **When generated:** This local request is generated when an application wants to read the user context information.
- **Effects of receipt:** If the user context has been set, the `ServiceManager` returns it to the application inside the field `CONTENT` of a `LocalMessage` object with the `TYPE` field set to `SUCCESS`. If the user context does not exist the `ServiceManager` replies to the application with a `LocalMessage` with the field `TYPE` set to `FAILED`.

6.2.4 Get Device Context

- **TYPE of LocalMessage:** `GET_DEV_CONTEXT`
- **Description:** This type of local request is used by the applications that want to retrieve the device context information.
- **Additional parameters:** none
- **When generated:** This local request is generated when an application wants to read the device context information.
- **Effects of receipt:** If the device context has been set, the `ServiceManager` returns it to the application inside the field `CONTENT` of a `LocalMessage` object with the `TYPE` field set to `SUCCESS`. If the device context does not exists the `ServiceManager` replies to the application with a `LocalMessage` with the field `TYPE` set to `FAILED`.

6.2.5 Get Application Context

- **TYPE of LocalMessage:** `GET_APP_CONTEXT`
- **Description:** This type of local request is used by the applications that want to retrieve its own context information.
- **Additional parameters:** none
- **When generated:** This local request is generated when an application wants to read its own context information.

- **Effects of receipt:** If the application context has been set, the `ServiceManager` returns it to the application inside the field `CONTENT` of a `LocalMessage` object with the `TYPE` field set to `SUCCESS`. If the application context does not exists the `ServiceManager` replies to the application with a `LocalMessage` with the field `TYPE` set to `FAILED`.

6.2.6 Get Remote Application Context

- **TYPE of LocalMessage:** `GET_REMOTE_APP_CONTEXT`
- **Description:** This type of local request is used by the applications thats want to retrieve the application context information of a remote node.
- **Additional parameters:** The ip address of the remote node of which the application wants to retrieve the application context.
- **When generated:** This local request is generated when an application wants to read an application context of a remote node.
- **Effects of receipt:** If the application context has been set for the given ip address, the `ServiceManager` returns it to the application inside the field `CONTENT` of a `LocalMessage` object with the `TYPE` field to `SUCCESS`. If the application context does not exists for the specified ip address, the `ServiceManager` replies to the application with a `LocalMessage` with the field `TYPE` set to `FAILED`.

6.2.7 Get Remote Contexts For a Given Community

- **TYPE of LocalMessage:** `GET_CONTEXTS_OF_COMMUNITY`
- **Description:** This type of local request is used by the applications that want to retrieve the context information of all the nodes belonging to a selected community.
- **Additional parameters:** A string representing the unique identifier of the community for which the application wants to retrieve the context information.
- **When generated:** This local request is generated when an application wants to read the context data related to all the nodes of a given community. This situation usually occurs when the application wants to evaluate the utility function for a given application content with respect to the interests of the nodes of a specific community.
- **Effects of receipt:** If the given community identifier is valid and the set of contexts related to that community inside CAMEO is not empty, the `ServiceManager` returns the related context information to the application in the field `CONTENT` of a `LocalMessage` object with the `TYPE` field set to `SUCCESS`. If there is no information inside CAMEO regarding the given community, the `ServiceManager` replies to the application with a `LocalMessage` with the field `TYPE` set to `FAILED`.

6.2.8 Evaluate The Utility For One or More Application Contents

- **TYPE of LocalMessage:** `EVALUATE_UTILITY`
- **Description:** This type of local request is used by the applications that want to obtain an evaluation of the utility function for one or more application contents with respect to the interests of the nodes of a previously encountered community.
- **Additional parameters:** A List of ids related to the application contents for which the application wants to evaluate the utility, a utility function expressed by a list of `ContentEvaluator` and an integer specifying the preferred social-oriented policy.
- **When generated:** This local request is generated when an application wants to obtain an evaluation of the utility function for one or more application content.
- **Effects of receipt:** The `ServiceManager` asks the `ContextManager` to calculate the result of the utility function for the given content ids. The content properties are retrieved from the remote contexts stored inside CAMEO. The `ContentEvaluator` is used to match the properties of the given contents with the preferences of the user, found inside the local user context. To better understand how the `ContextManager` calculates the utility of the given contents see Section 5.6. If the contents ids passed by the requester application are valid and are known to CAMEO, the `ServiceManager` replies to the application with a list containing the results of the utility function evaluations indexed by the content id. The result is placed in the field `CONTENT` of a `LocalMessage` object with the `TYPE` field to `SUCCESS`. If the content ids are not valid or unknown to CAMEO, the `ServiceManager` replies to the application with a `LocalMessage` with the field `TYPE` set to `FAILED`.

6.3 Send a Message Over The Network

- **Name of the service primitive:** `SendMessage`
- **Description:** This method allows an application to send a customized message over the network to another node running an application using the same communication port as the sender.
- **Type:** Confirmed, Asynchronous, Remote
- **Semantics:** The primitive shall provide the following parameters:
  1. *packet* (input)
     - Type: `LocalMessage`
     - Accepted values: a valid LocalMessage with the boolean field `NET_MESSAGE` set to `TRUE` and the field `PAYLOAD` set with the content which the application wants to send over the network.
     - Description: This is the content of the message to be sent over the network.

2. *dest* (input)
   - Type: ip address
   - Accepted values: any valid ip addresses
   - Description: This is the ip address of the destination.
3. *broadcast* (input)
   - Type: boolean
   - Accepted values: { true,false }
   - Description: Whether or not the packet must be sent to all the nodes in the 1-hop area (like a beacon)
4. *result* (output)
   - Type: boolean
   - Accepted values: { true,false }
   - Description: True if the message is successfully processed by the `TransmissionManager`. False in case of errors.

- **When generated:** This primitive is generated when an application wants to send a message to one or more nodes in the network. It is typically used to exchange application contexts.
- **Effects of receipt:** The receipt of this primitive by the `ServiceManager` causes CAMEO to pass the message to the `ForwardingManager`, which calculates the best next hop for the delivery of the message and then passes the message to the `TransmissionManager`, which sends the message over the network toward the destination. If the `TransmissionManager` has not been started yet or is in idle state, a `ReliableMessageNotSentException` (or a `UnreliableMessageNotSentException` in case of a broadcast message) is thrown.

7 Callback Interface

Each application must extend `CallbackInterface`, implementing the methods called by CAMEO to notify events. During the registration procedure each application passes its custom `CallbackInterface` to CAMEO. CAMEO maintains a list of `CallbackInterface` in a dedicated data structure. An entry of this list is removed when the respective application logs out or crashes. The list is indexed by the application identifier. CAMEO sends two different types of notifications: (i) broadcast notifications (i.e., events to be sent to all the registered applications); (ii) direct notifications (i.e., events, errors or messages destined to a specific application). When CAMEO sends a notification, it passes to the applications a `LocalMessage` object, containing the details of the occurred event. In the following we provide a detailed description of the methods defined by the `CallbackInterface`.

7.1 Receive a Generic Asynchronous Event

- **Name of the method:** `onNewEvent`

- **Description:** This is a generic method used by CAMEO to notify different types of events.
- **When generated:** This method is invoked by CAMEO to notify the applications about an event occurred.
- **Parameters returned by CAMEO:** a *LocalMessage* containing the details of the notification. The different types of notifications are identified by the field `TYPE` of the returned *LocalMessage*. The types of event defined by CAMEO are the following:
  - *LINK_LOSS*: Generated by the `NetworkManager` when the status of the network interface changes from `CONNECTED` to `DISCONNECTED`. No extra parameters.
  - *LINK_RESUME*: Generated by the `NetworkManager` when the status of the network interface changes from `DISCONNECTED` to `CONNECTED`. No extra parameters.
  - *NEIGHBOR_IN*: Generated by the `ContextManager` when a new neighbor joins the 1-hop area. The ip address of the new neighbor is passed inside the `payload`.
  - *NEIGHBOR_OUT*: Generated by the `ContextManager` when a known neighbor leaves the 1-hop area. The ip address of the neighbor is passed inside the `payload`.
  - *COMMUNITY_CHANGED*: Generated by the `ContextManager` when the actual community changes. The unique identifier of the new community is passed inside the `payload`.
  - *NEIGHBOR_CONTEXT_UPDATED*: Generated by the `ContextManager` when new information regarding a neighbor context is available. This information can refer to both user and application contexts. Thanks to this notification the applications can discover new contents available for downloading from remote nodes. The ip address of the neighbor and the version numbers related to its contexts, received from the `Beaconing` module, are passed to the application inside the `payload` object.

7.2 Receive a Message From a Remote Node

- **Name of the method:** `onReceivedMessage`
- **Description:** This notification informs the applications about the presence of an incoming message from a remote node.
- **When generated:** This method is invoked by CAMEO when it receives a message for a specific application from a remote node.
- **Parameters returned by CAMEO:** a *LocalMessage* containing the content of the message and the ip address of the sender.

8 Conclusions

The main novelty of CAMEO is a light-weight context-aware middleware platform to allow easy and efficient development of MSN applications in opportunistic networks. In this paper we provided a detailed description of CAMEO architecture and the technical specification of its software components. The prototype has been tested and evaluated through an experimental testbed. Performance results showed the efficiency of CAMEO in collecting and managing a multidimensional context and supporting the development of a real example of MSN applications dedicated to tourists. Currently we are extending CAMEO in several directions. We are analyzing the applicability of a context model based on CML language to improve reasoning capabilities of CAMEO and identify possible extensions in the context definition. Concurrently, we are integrating SensorML standard in the Device Context Provider module in order to implement a uniform data format for sensors' readings. This will allow CAMEO to provide additional functionalities for the automatic characterization of a node context based on sensor readings.

References

[1] Android developer guide. URL `http://developer.android.com/guide/topics/fundamentals.html`

[2] C++ corba idl. URL `http://www.omg.org/spec/CPP/1.2`

[3] Cyanogenmod 6.1.0-n1 firmware. URL `http://wiki.cyanogenmod.com/index.php?title=CyanogenMod_6_Changelog`

[4] Dalvik java virtual machine. URL `http://code.google.com/p/dalvik/`

[5] Java corba idl. URL `http://www.omg.org/spec/I2JAV/1.3`

[6] Abdelzaher, T., Anokwa, Y., Boda, P., Burke, J., Estrin, D., Guibas, L., Kansal, A., Madden, S., Reich, J.: Mobiscopes for human spaces. IEEE Pervasive Computing **6**(2), 20–29 (2007). DOI http://doi.ieeecomputersociety.org/10.1109/MPRV.2007.38

[7] Arnaboldi, V., Conti, M., Delmastro, F.: Implementation of CAMEO : a Context-Aware MiddlEware for Opportunistic Mobile Social Networks. In: 12th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WOWMOM), BEST DEMO AWARD (2011)

[8] Boldrini, C., Conti, M., Passarella, A.: ContentPlace: social-aware data dissemination in opportunistic networks. In: ACM MSWiM, pp. 203–210. ACM, Vancouver, British Columbia, Canada (2008). DOI http://doi.acm.org/10.1145/1454503.1454541

[9] Boldrini, C., Conti, M., Passarella, A.: Context and resource awareness in opportunistic network data dissemination. In: IEEE AOC Workshop (2008)

[10] Boldrini, C., Conti, M., Passarella, A.: Exploiting users social relations to forward data in opportunistic networks: The HiBOp solution. Pervasive and Mobile Computing **4**(5), 633–657 (2008)

[11] Boldrini, C., Passarella, A.: Hcmm: Modelling spatial and temporal properties of human mobility driven by users' social relationships. Computer Communications **33**(9), 1056–1074 (2010)

[12] Borgia, E., Conti, M., Passarella, A.: Autonomic detection of dynamic social communities in opportunistic networks. In: Med-Hoc-Net, pp. 142–149. IEEE (2011)

[13] Botts, M., Robin, A.: Open Geospatial Consortium Inc . Sensor Model Language (SensorML) Implementation Specification (2007)

[14] Conti, M., Das, S.K., Bisdikian, C., Kumar, M., Ni, L.M., Passarella, A., Roussos, G., Tröster, G., Tsudik, G., Zambonelli, F.: Looking ahead in pervasive computing: Challenges and opportunities in the era of cyber-physical convergence. Pervasive and Mobile Computing **8**(1), 2–21 (2012)

[15] Conti, M., Kumar, M.: Opportunities in Opportunistic Computing. Computer **43**(1), 42–50 (2010). DOI 10.1109/MC.2010.19

[16] Danon, L., Díaz-Guilera, A., Duch, J., Arenas, A.: Comparing community structure identification. Journal of Statistical Mechanics: Theory and Experiment **9**, 10 (2005)

[17] Hui, P., Crowcroft, J., Yoneki, E.: Bubble rap: Social-based forwarding in delay tolerant networks. In: the 9th ACM international symposium on Mobile ad hoc networking and computing, pp. 241–250. ACM (2008)

[18] Newman, M.E.J.: Detecting community structure in networks. The European Physical Journal B-Condensed Matter and Complex Systems **38**(2), 321–330 (2004)

[19] Pelusi, L., Passarella, A., Conti, M.: Opportunistic networking: data forwarding in disconnected mobile ad hoc networks. Communications Magazine, IEEE **44**(11), 134–141 (2006)

[20] Percivall, G., Reed, C., Davidson, J.: Open Geospatial Consortium Inc . Sensor Web Enablement : Overview And High Level Architecture . (2007)

[21] Trifunovic, S., Distl, B., Schatzmann, D.: WiFi-Opp: ad-hoc-less opportunistic networking. Proceedings of the 6th (2011). URL `http://dl.acm.org/citation.cfm?id=2030664`

[22] Yoneki, E., Hui, P., Chan, S.Y., Crowcroft, J.: A socio-aware overlay for publish/subscribe communication in delay tolerant networks. In: the 10th ACM Symposium on Modeling, analysis, and simulation of wireless and mobile systems. ACM (2007)